Amendments to the Specification:

Please replace the paragraph beginning on page 3, line 21 with the following amended paragraph:

As known in the art the concept of a state is a way of defining a related group of graphics primitives; that is, a set of primitives having a common attribute or need for a particular type of processing define a single state. For example, if an object to be rendered on a display comprises multiple types of textures, graphics primitives corresponding to each type of texture comprise a separate state. A given state may be realized through state data. For example, the DirectX 8.0 standard promulgated by Microsoft Corporation defines the functionality for so-called programmable vertex shaders (PVSs). A PVS is essentially a generic video graphics processing platform, the operation of which is defined at any moment according to state data.

a-1

Please replace the paragraph beginning on page 5, line 23 with the following amended paragraph:

The present invention provides a technique for maintaining and using multiple sets of state data in state-related buffers. In particular, up to N [[states]] sets of state data are stored in a buffer such that a total length of the N sets of state data does not exceed the total length of the buffer. While stored in the buffer, at least one of the N sets of state data may be used to process graphics primitives. When it is desired to add an additional set of state data, it is first determined whether a length of the additional set of state data would exceed available space in the buffer. When the length of the additional set of state data would exceed the available space in the buffer, storage of the additional set of state data in the buffer is delayed until at least M of the N sets of state data are no longer being used to process graphics primitives, wherein M is less than or equal to N. The M sets of state data are preferably those sets of state data that would be at least partially overwritten by the additional set of state data. Where the buffer is implemented as a ring buffer, this technique allows state data to be continuously updated in a

a2 cnt

single buffer while minimizing the impact of state data updates. In another embodiment of the present invention, additional sets of state data are prevented from being added to the buffer if a maximum number of allowed states is already stored in the buffer. In this manner, the present invention ensures that state data will not be corrupted when additional state data is to be added to the buffer.

Please replace the paragraph beginning on page 9, line 19 with the following amended paragraph:

As shown in FIG. 3, the buffer 303 comprises N sets of state data stored sequentially. An amount of available space is also illustrated in the buffer 303 and comprises locations in the buffer 303 not otherwise occupied by the N sets of state data. In a preferred embodiment, the buffer 303 is implemented as a ring buffer. Ring buffers are well known to those having ordinary skill in the art, and need not be described in further detail herein. Based on the example illustrated in FIG. 3, the PVS engine 202 can operate in accordance with any of the sets of state data, labeled [[1-N]] 1 through N. Because any one of these sets of state data can be loaded while the PVS engine 202 is executing in accordance with another set of state data, the latencies encountered in prior art systems are avoided.

Please replace the paragraph beginning on page 12, line 14 with the following amended paragraph:

Referring now to FIG. 7, there is illustrated a flow chart describing operation of the present invention. In particular, two parallel paths of processing are illustrated in FIG. 7. On the left, comprising blocks 702-[[714]] 710, processing implemented by a host (state data source) is shown. In a preferred embodiment, the state data source is embodied by a computer-implemented application providing data to a driver that, in turn, provides the state data to the programmable vertex shader. All processing of vertices for a given set of primitives is also

initiated by the computer-implemented application and driver. The driver is preferably implemented as instructions stored in virtually any type of computer-readable memory, such as memory 108 in FIG. 1. On the right of FIG. 7, processing performed by a programmable vertex shader is illustrated by blocks ~~718-726~~ 720-730.

Please replace the paragraph beginning on page 14, line 19 with the following amended paragraph:

Furthermore, a flush command may be sent to the PVS at any time prior to overwriting currently-stored state data in a state data buffer. That is, if it is determined that an additional set of state data would prematurely overwrite a portion of the state data buffer, the flush command could be sent before any of the additional [[set]] sets of state data is sent. Alternatively, an amount of the additional set of state data not exceeding the currently available space in the buffer could be first sent to the PVS for storage in the buffer. Then, at any time prior to overwriting a currently-used state data buffer location, the flush command could be sent thereby preventing any subsequent writes to the state data buffer until the requisite number of state data sets are no longer being used. Thereafter, the remaining portion of the additional set of state data could be stored in the buffer. In this manner, the delay associated with loading the additional set of state data could be reduced even further.

Please replace the paragraph beginning on page 16, line 13 with the following amended paragraph:

When it is determined that a less than the maximum number of states are currently stored in the buffer, processing continues at block 724 where it is determined whether a flush command has been encountered. Note that the decisions of blocks 720 and 724 have been illustrated in a serial fashion for convenience of explanation. That is, although the decisions of blocks 720 and 724 have been illustrated in FIG. 7 as occurring in a specific order, in practice, the decisions

illustrated by blocks 720 and 724 may occur asynchronously relative to each other. If a flush command has been received, processing continues at step 726 where it is determined whether the number of sets of state data required to satisfy the flush command are no longer being used. For example, in the preferred embodiment, the flush command requires that all currently stored states be completed. However, as described above, a more flexible flush command may be implemented in which the particular number of sets of state data to be completed may be specified. Regardless, if the required number of sets of state data are not completed (i.e., they are still in use), processing continues at block 728 where the PVS awaits de-allocation of the required number of sets of state data. Once de-allocation has occurred, or where a flush command is not encountered, processing continues at block 730 where the state data is written to the buffer.